Higher-Order Glitch Resistant Implementation of the PRESENT S-Box

Thomas De Cnudde[1], Begül Bilgin[1,2], Oscar Reparaz[1], and Svetla Nikova[1]

[1] KU Leuven, ESAT-COSIC and iMinds, Belgium
{name.surname}@esat.kuleuven.be
[2] University of Twente, EEMCS-SCS, The Netherlands

Abstract. Glitches, occurring from unwanted switching CMOS gates, have been shown to leak information even when side-channel countermeasures are applied to hardware cryptosystems. The polynomial masking scheme presented at CHES 2011 by Roche *et al.* is a method that offers provable security against side-channel analysis at any order even in the presence of glitches. The method is based on Shamir's secret sharing and its computations rely on a secure multi-party computation protocol. At CHES 2013, Moradi *et al.* presented a first-order glitch resistant implementation of the AES S-box based on this method. Their work showed that the area and speed overheads resulting from the polynomial masking are high. In this paper, we present a first-order glitch resistant implementation of the PRESENT S-box which is designed for lightweight applications, indicating less area and randomness requirements. Moreover, we provide a second-order glitch resistant implementation of this S-box and observe the increase in implementation requirements.

Keywords: Polynomial masking, Glitches, Sharing, PRESENT, S-box

1 Introduction

Radio frequency identification (RFID) systems, wireless sensor networks, smart cards and other compact mobile applications have become prevalent in everyday life. Their widespread deployment in applications ranging from supply chains to intelligent homes and even electronic body implants, has made their security a pressing issue. While block ciphers provide sufficient security against cryptanalysis for these applications, their hardware implementations are susceptible to side-channel leakage. By exploiting these leaks through side-channel analysis (SCA), a cryptosystem can be compromised more easily than promised by the cryptanalytic security. A common side-channel analysis is Differential Power Analysis (DPA) [14]. DPA exploits dependencies between the instantaneous power consumption of a device and the intermediate values arising in the computation of a cryptographic operation.

Several countermeasures have been proposed to cope with these side-channels. Secure logic styles that balance the power consumption of different data values [24] can be used or noise can be increased in the form of random delays,

random execution orders or by inserting dummy operations [25]. Even though an analysis becomes harder as this noise increases, these techniques do not provide provable security. A popular countermeasure that does provide provable security under certain assumptions is *masking* [5, 10]. This method conceals *sensitive* information, such as key and plaintext related information, using random values. Compared to a naive implementation, a well implemented masked implementation typically offers more resistance against power analysis attacks, and makes the attack much more expensive as the order d of the masking increases. This masking order d in turn defines the order $d + 1$ of the attack needed to retrieve the sensitive information. This attack order sets the number of shares that are jointly exploited by either analyzing the $(d + 1)^{th}$-order statistical moment of the leakage at one point in time or by nonlinearly combining leakages from $d + 1$ points in time. Such an attack is known as a $(d + 1)^{th}$-order DPA attack. A d^{th}-order secure implementation can consequently always be broken by a $(d + 1)^{th}$-order attack. When the attack order is larger than one, this is known as a higher-order DPA (HO-DPA) attack [5, 17].

Masking is however deteriorated by the switching behaviour of CMOS transistors, the so called glitching effect [15, 16]. Two masking schemes that show provable security against DPA in the presence of glitches, or glitch resistance for short, are the *polynomial masking scheme* [22] and *threshold implementations* [19]. While, at the time of writing, the latter achieves glitch resistance at the first-order only, the former provides this security also for higher orders. Therefore, we consider the polynomial masking scheme in this paper.

Masking introduces an overhead on the area and throughput. To avoid overly large and slow implementations, we will focus on lightweight, *i.e.* compact and power efficient, block ciphers. A popular lightweight block cipher is PRESENT [3] which, as of 2012, is part of the ISO/IEC 29192-2 standard [13], making its side-channel resistance relevant. Besides PRESENT, its S-box is also used in other lightweight cryptographic algorithms, including the LED block cipher [12], the GOST revisited block cipher [20] and the PHOTON lightweight hash function [11]. In this paper, we focus on glitch resistant implementations of the nonlinear part of PRESENT, the S-box, since this is typically the most challenging part of a masked implementation.

Related work. An algorithmic description of a first-order glitch resistant Advanced Encryption Standard (AES) implementation using the polynomial masking scheme is given in [22]. In [18], this description is used to implement a first-order glitch resistant AES S-box on an FPGA. The PRESENT S-box has, to our knowledge, not yet been implemented using polynomial masking.

Contribution. In this paper, we present a first- and a second-order polynomially masked implementation of the 4-bit PRESENT S-box. To our knowledge, this is the first second-order PRESENT S-box implementation showing resistance against second-order DPA in the presence of glitches. The implementations are based on the guidelines for the first-order glitch resistant AES implementation

proposed in [18]. We also present experimental confirmation showing that the implementations indeed achieve their claimed security. To this end, we applied univariate and bivariate leakage detection based on *Welch's t-test*.

Organization. Section 2 introduces the necessary background regarding the polynomial masking scheme and the PRESENT S-box.The design decisions, hardware implementations and their costs are presented in Section 3. The SCA results are shown in Section 4. Finally, the conclusion is drawn in Section 5.

2 Preliminaries

2.1 PRESENT Block Cipher

The PRESENT block cipher [3] is a symmetric key encryption algorithm designed considering the heavy constraints on performance, area and timing requirements of lightweight hardware applications. Its block length equals 64-bits. Key lengths of 80- and 128-bits are supported, which are referred to as PRESENT-80 and PRESENT-128 respectively. For lightweight applications, PRESENT-80 is recommended. The PRESENT cipher performs 31 rounds followed by a final key whitening stage. Each round consists of a binary addition with the round key and a substitution-permutation network. The permutation layer is bit oriented and can easily be implemented by wiring, making it very hardware friendly. The substitution layer applies 16 identical 4-bit S-boxes governed by the following table:

Table 1. 4-bit to 4-bit substitution of the PRESENT S-box [3] in hexadecimal notation.

z	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[z]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

2.2 Polynomial Masking Scheme

Side-channel resistance in the presence of glitches can be achieved at any order by the polynomial masking scheme [22]. Sensitive variables are masked using Shamir's secret sharing scheme [23] and computations on the resulting shares are performed using the *BGW's* secure multi-party computation protocol [2].

In Shamir's scheme, a secret $Z \in K \equiv \mathbb{F}_{2^m}$ is shared among $n < 2^m$ players such that d+1 players are needed to reconstruct Z. To this end, a *dealer* generates a degree-d polynomial $P_Z(X) \in K[X]$ with constant term Z and secret, random coefficients a_i:

$$P_Z(X) = Z + \sum_{i=1}^{d} a_i X^i$$

When working in the field K, we will denote binary addition and field multiplication by + and . respectively.

This polynomial is then evaluated in n distinct, non-zero elements $\alpha_1, ..., \alpha_n \in K$, which are called the *public coefficients* and are available to all players. Lastly, each resulting value $Z_i = P_Z(\alpha_i)$ is distributed to its corresponding player i. The secret Z can be reconstructed using the first row $(\lambda_1, ..., \lambda_n)$ of the inverse of the $(n \times n)$ Vandermonde matrix $(\alpha_i^j)_{1 \leq i,j \leq n}$ as:

$$Z = \sum_{i=1}^{n} \lambda_i Z_i$$

This is exemplified for the second-order in Appendix B.

BGW's protocol defines how to securely operate on the shares. We can distinguish between operations that can be processed by all players independently and operations that need communication between the players. Multiplication of a share and a constant, addition of a share and a constant and addition of two shares can be processed by each player independently. As a result, these operations can be implemented straightforwardly. Multiplication of two shares, which is referred to as *shared multiplication*, requires the players to exchange information, which complicates its secure execution. This operation has to be performed in three steps [22]:

1. Each player multiplies its shares, resulting in a $2d$-degree polynomial
2. Each player masks the result of the previous multiplication and sends these shares to all other players
3. Each player reconstructs the result by interpolation and evaluation in the public coefficients

When the square of a share is desired, the shared multiplication can be omitted when following conditions are imposed [22]:

- The public coefficients α_i are distinct and non-zero
- The public coefficients α_i are stable over the Frobenius automorphism: for every α_i, there exists an α_j such that $\alpha_j = \alpha_i^2$

Each player can then independently perform the squaring on its own share but a reordering of the shares is needed between player i and player j when $i \neq j$ to keep the right public coefficient linked to its corresponding player.

To achieve glitch resistance with this masking scheme, two conditions need to be fulfilled. Firstly, the number of players has to exceed twice the degree of the polynomial, *i.e.* $n > 2d$. Secondly, each player has to leak independently of all other players.

2.3 Cyclotomic Classes

The masking complexity of an S-box is defined in [4] as the minimal number of nonlinear multiplications required to evaluate its polynomial. These nonlinear multiplications correspond to shared multiplications.

When calculating a power x^α from another power x^β, a nonlinear multiplication can be omitted if and only if α and β lie in the same *cyclotomic class*. A cyclotomic class is defined as follows.

Definition 1 (Cyclotomic Class). *With $m \in \mathbb{N}$ and $\alpha \in [0; 2^m - 2]$, the cyclotomic class C_α of α w.r.t. m is defined as:*

$$C_\alpha = \{\alpha \cdot 2^i \bmod 2^m - 1, \quad i \in [0; m-1]\}$$

For the PRESENT S-box, we work in field $\mathbb{F}_{2^4}$. Its corresponding cyclotomic classes are:

$$C_0 = \{0\}, C_1 = \{1, 2, 4, 8\}, C_3 = \{3, 6, C, 9\}, C_5 = \{5, A\}, C_7 = \{7, E, D, B\} \tag{1}$$

An important property is that we can cycle through the elements of a cyclotomic class by squaring, which can be performed independently by all players when the conditions listed in Section 2.2 are fulfilled. As squaring is linear in $\mathbb{F}_{2^4}$, the S-box complexity equals the number of different transitions between these classes required to evaluate the S-box substitution function.

3 Hardware Implementation

In this section, the hardware implementations of the first-order and second-order glitch resistant PRESENT S-box are explained. First, the polynomial of the S-box and its evaluation order are established. Then the detailed first-order glitch resistant implementation is discussed. Afterwards, the modifications required to achieve second-order glitch resistance are given. This section is concluded with an overview of the implementation requirements.

3.1 Evaluation Order

The substitution of any 4-bit S-box can be expressed as a unique polynomial over $\mathbb{F}_{2^4}$ with a degree of at most $2^4 - 1 = 15$. This polynomial can be obtained by expanding the following expression [7]:

$$S(x) = \sum_{z \in \mathbb{F}_{2^4}} S(z)(1 + (x + z)^{15}) = \sum_{i=0}^{15} c_i x^i$$

Using the Mattson-Solomon polynomial, the coefficients c_i of $S(x)$ can directly be computed by:

$$c_i = \begin{cases} S(0), & \text{if } i = 0 \\ \sum_{k=0}^{2^4-2} S(\alpha^k)\alpha^{-ki}, & \text{if } i \leq i \leq 2^4 - 2 \\ S(1) + \sum_{i=0}^{2^4-2} c_i, & \text{if } i = 2^4 - 1 \end{cases}$$

where α is a primitive element in $\mathbb{F}_{2^4}$.

If we use $x^4 + x + 1$ as irreducible polynomial for the construction of $\mathbb{F}_{2^4}$, we get the following polynomial for the PRESENT S-box given in Table 1.

$$\begin{aligned} S(x) = {} & Dx^{14} + Dx^{13} + Cx^{12} + Ex^{11} + 9x^{10} + 9x^9 + 7x^8 \\ & + 4x^7 + Cx^6 + Ax^5 + Ex^4 + 7x^3 + 7x^2 + C \end{aligned}$$

The evaluation order of this polynomial is an adaptation of the proposal by Carlet *et al.* in [4] to reduce the required memory and area by processing sequentially instead of in parallel. The block diagram of the PRESENT S-box evaluation is depicted in Figure 1. The gray multipliers symbolize a field multiplication with a constant, while the black multipliers represent a shared multiplication. Starting from input x, squaring is consecutively carried out until all elements of the cyclotomic class C_1 from Equation (1) are covered. The last element of that class is then multiplied with x to access cyclotomic class C_3, where all elements are again obtained by squaring. After a multiplication with x, squaring is performed again to reach all elements in C_7. To access the final cyclotomic class C_5, a multiplication with x^{11} is chosen, as multiplying our last obtained power with x would lead back to class C_1. This value will need to be stored separately.

From this discussion it is apparent that a shared multiplier cannot be omitted. As our primary design goal is low area, we choose to only implement a shared multiplier to handle all shared multiplications. However, by evaluating the polynomial this way, the designs can easily be extended with a dedicated squaring circuit and benefit from a significant reduction of required randomness. This extension is left as future work.

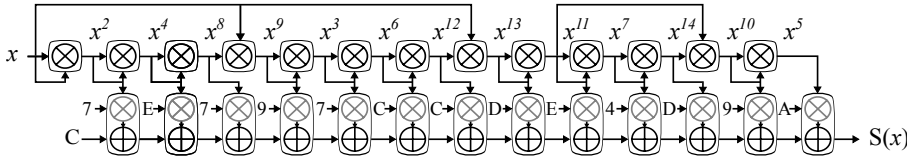

Fig. 1. Block diagram of the evaluation for the PRESENT S-box.

3.2 First-Order Glitch Resistant PRESENT S-box

To achieve first-order glitch resistance, both conditions in Section 2.2 have to be fulfilled. Namely, our sensitive variables need to be masked by a first-order polynomial and need to be shared between three players with independent side-channel leakage. In order to achieve this independent leakage, we choose to temporally separate the players' operations. After each operation, the intermediate results are stored and left unaltered while another player is active. The design is shown in Figure 2 and is similar to the AES S-box implementation from [18]. This design is compatible with all combinational finite field multipliers. The one used in our implementations is given in Appendix A.

Shared multiplier. As pointed out in Section 2.2, the shared multiplication differs from the other operations in that it needs communication between the players. To achieve this, the computations are divided in two parts and the communicated intermediate values are stored in registers.

Step 1 and Step 2 (Section 2.2) of the shared multiplication are performed in the MULT_EL1$_i$ blocks. With every shared multiplication, each player receives

a new random coefficient a_i to remask the multiplication of its input shares t_i. The reconstruction in Step 3 is handled by the MULT_EL2$_i$ blocks once all intermediate results are available.

The detailed working principle is described in *series* of clock cycles. Such a series consists of six clock cycles and is related to the control signals $em_{1 \leq i \leq 6}$, which can be seen in Figure 2. During each series, a shared multiplication is realized.

- The first clock cycle of a series, enables signal em_1. The two required inputs for the shared multiplier are selected by $selm_1$. At the same time, a new random number a_1 is fed to the MULT_EL1$_1$ block. Together with this random number, the fixed public coefficients α_1, α_2 and α_3 are used to remask the multiplied input shares t_1.
- The same procedure is repeated on the second clock cycle using signal em_2 in block MULT_EL1$_2$ and on the third clock cycle using em_3 in block MULT_EL1$_3$. After the third clock cycle, all intermediate results are available.
- In the fourth clock cycle, by activating signal em_4, the intermediate results related to the first public coefficient α_1 are stored in the registers $q_{1,1}$, $q_{2,1}$, $q_{3,1}$. The combinatorial logic in block MULT_EL2$_1$ then performs the reconstruction using λ_1, λ_2 and λ_3. This outputs the first share of the shared multiplication. The result is not saved in this clock cycle, but will be done at the start of the next series, with the activation of the select signal em_1.
- In the fifth and sixth clock cycles, the same principles as in the fourth clock cycle apply. The enable signal em_5 handles the reconstruction related to the second public coefficient α_2 in block MULT_EL2$_2$ and em_6 serves the reconstruction related to the third public coefficient α_3 in block MULT_EL2$_3$.

Note that, except for the registers, the shared multiplier is entirely combinatorial. Therefore, the MULT_EL1$_i$ and MULT_EL2$_i$ blocks are only active when a new value is assigned to their input registers. After one clock cycle, the intermediate values reach their stable states and the blocks stay idle until their input registers are changed again. By temporally separating the em_i signals with a carefully designed control unit, we achieve the required temporal separation.

Input selection. The right inputs for the shared multiplier are selected by the multiplexers in the CTRL_EL$_i$ blocks. A glitch on the select signal of a multiplexer can temporarily change the inputs of the shared multiplier and induce processing in a player that is supposed to be idle. This would result in an overlap of leakages of different players and would eradicate the temporal separation. To avoid this, these $selm_i$ signals are synchronised. As was noted in Section 3.1, we need to store one extra intermediate value x^{11}. When the shares of this value are output at the MULT_EL2$_i$ blocks, the es_1, es_2 and es_3 signals follow the levels of the em_1, em_2 and em_3 signals to store the shares of x^{11} in separate registers.

Addition and accumulation. To calculate the polynomial, the powers of x need to be multiplied with a constant and accumulated with the previously

obtained results. This is handled by the ADD_ACC_EL$_i$ blocks. When the shares of a desired power of x are ready at the outputs of the shared multiplier, the ea_i signals activate with the corresponding em_i signals. With the activation of an ea_i signal, a new coefficient chosen by *selcoeff* is fed to an input of the ADD_ACC_EL$_i$ multiplier, resulting in the right multiplication of a constant and its corresponding power of x. In the first series of clock cycles, the constant value C of the polynomial is added to an empty register using the $sela_i$ signal, which activates with its corresponding em_i signal. In all following series of clock cycles, the register output is chosen to accumulate the results. The eo_i signal enables the output share of player i when the register holds the final value. This signal also activates with its corresponding em_i signal.

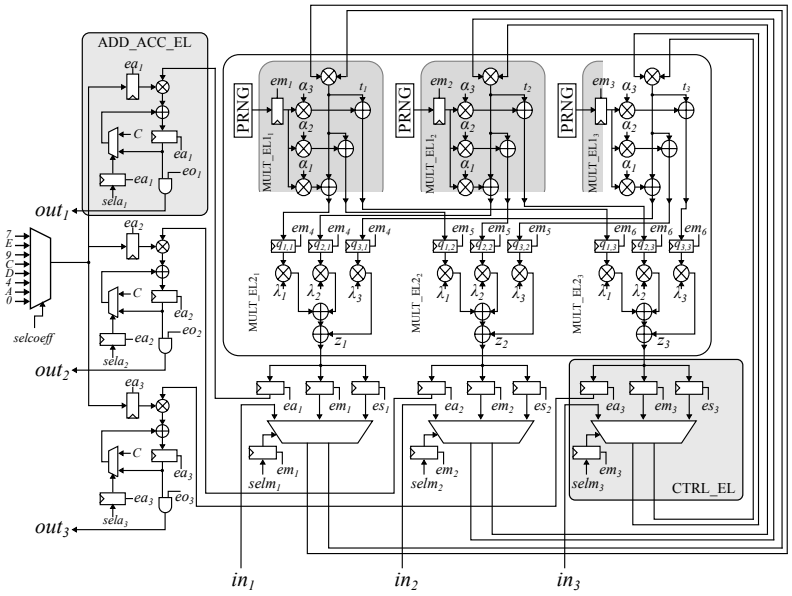

Fig. 2. Architecture diagram for the first-order PRESENT implementation.

3.3 Second-Order Glitch Resistant PRESENT S-box

We will now discuss how to extend our first-order design to the second-order. Again, both conditions in Section 2.2 need to be fulfilled. To provide second-order glitch resistance, our sensitive variables are now masked using a second-order polynomial and shared among five players. We again choose temporal separation to decouple the leakages of the different players. The operations in this (5,2)-sharing scheme are detailed in Appendix B. Figure 5 in Appendix C shows the resulting architecture diagram.

Shared multiplier. The MULT_EL1$_i$ blocks now require two instead of one random coefficients to mask the multiplication of the inputs. Furthermore, the

evaluation of the polynomial is done in five public coefficients and their squared value is needed. When hardcoding the public coefficients and their squares, we additionally require seven multiplications, seven additions and one register. Each player now requires five instead of three registers to share the intermediate results. The MULT_EL2$_i$ blocks need two extra multiplications and two extra additions to perform the reconstruction in the (5,2)-sharing scheme.

The control schedule is changed to incorporate five players. The same principles from Section 3.2 apply, but we need 10 em_i signals, the first five to control the MULT_EL1$_i$ blocks and the last five to store the intermediate values in the registers.

Input selection, addition and accumulation. The only change made in these operations is the extension from three CTRL_EL (resp. ADD_ACC_EL) blocks to five.

The security against second-order DPA in the presence of glitches of this implementation can theoretically be explained as follows. As a second-order polynomial is used to divide the shares among five players, the shares of at least three players are required to interpolate the masked secret. Mixing up to two observations of intermediate variables will therefore not lead to enough information to reveal the secret variable. Furthermore, as the computations of each player are temporally separated, the information leaked by glitches is contained to the share of that player only and is not influenced by the shares of other players. This theoretical proof is valid for all orders when appropriate changes to the players are considered.

3.4 Implementation Requirements

The total area in NAND gate equivalents (GEs) covers 3594 GE and 8338 GE for the first- and second-order glitch resistant implementation respectively. The largest contributions come from the shared multiplier (37.8% and 59.6%) and the control unit (41.8% and 25.7%), both for the first- and second-order respectively). The detailed area requirements of the different blocks are given in Table 3 in Appendix D. The results are obtained from Synopsys 2010.03 using the NanGate 45nm Open Cell Library [1].

The first-order implementation requires 89 clock cycles from the activation of the request signal to the output of all shares. For the second-order implementation, this number becomes 149 clock cycles. The secure evaluation of the first-order PRESENT S-box requires 156-bits of randomness. If a squaring module is used, this randomness can drop to 36-bits trading off area. For secure evaluation of the second-order PRESENT S-box, the required randomness changes to 520-bits (resp. 120-bits when a squaring module is used). As all public coefficients should be distinct and non-zero, up to 15 players can be accommodated. By imposing the condition that $n > 2d$, this leads to a maximum of a seventh-order glitch resistant implementation for the PRESENT S-box. For all possible orders d of glitch resistance, the required number of randomness and clock cycles are summarized in Table 2.

Table 2. Number of clock cycles and randomness required for a d^{th}-order glitch resistant PRESENT S-box implementation.

Number of Clock Cycles	$30(2d+1)-1$
Randomness (bits)	$52d(2d+1)$
Randomness with squaring module (bits)	$12d(2d+1)$

4 SCA evaluation

In this section we provide experimental evidence that our implementations provide a reasonable guarantee against typical power analysis attacks. We perform leakage detection tests on the PRESENT S-box, implemented on a SASEBO-G board [21]. The board is externally clocked with a stable, relatively low-frequency clock source of 3.072MHz. All the randomness required for the computations is generated by an AES-based PRNG on the control FPGA. All the tests were performed with $1M$ traces unless explicitly stated otherwise.

For our evaluation, we use the non-specific fixed-vs-random methodology of [6, 9]. In a nutshell, the leakage detection test assesses whether the means of power consumption traces, conditioned on any intermediate, are equal or not. In the context of first-order masking, this means whether the masking is sound or not. We stress that by using a non-specific test, we are targeting all intermediates appearing during the computation of an S-box. This allows us to test the implementation against a wide range of leakages, without assuming how the implementation may leak.

The original methodology starts by taking two sets of measurements corresponding to fixed plaintext and random plaintext. Then, a hypothesis test is applied time sample per time sample to test whether the means of the two populations are the same or not. Normally, a Student T-test is applied. Having set a significance level beforehand, the result of the test is directly interpretable in terms of probability. In our case, a value of the t-test statistic beyond 4.5 means that there is leakage with high probability. For details on the test, we refer to Appendix E. For our purposes of testing the higher-order security, we adapt the methodology to analyze higher-order moments in univariate and bivariate distributions (two time samples jointly analyzed). This is achieved by preprocessing the power traces through a suitable combination function. In our case, we use the centered product.

We begin with a univariate analysis of the first-order protected implementation. As a first sanity check of our experimental setup, we performed a univariate first-order test with the PRNG switched off, thus deliberately disabling the masking. The result of the t-test on the unmasked first-order implementation is given in Figure 6 in Appendix E. This clearly shows that the implementation is leaking since the t-test statistic trace exceeds the confidence threshold $C = \pm 4.5$ in several clock cycles, which is expected as the masking is inactive. If we repeat the experiment with the PRNG enabled, the t-test statistic never exceeds the predefined threshold as the top left corner of Figure 3 indicates.

We repeated the test on centered and squared traces. This is equivalent to test whether there is information leakage on the variances. Note that the first-order protected implementation is expected to leak in the second moment, as Figure 3 indicates. This only provides us with the evidence that we indeed have enough traces to show that the first-order attack is more expensive in terms of traces than higher-order ones, and thus our goal of first-order security is attained.

We proceeded with a univariate analysis of the second-order glitch resistant implementation. The process follows the lines of the first-order protected implementations and the results are again shown in Figure 3. We can see that the implementation is indeed first- and second-order univariate secure up to $1M$ traces. The implementation leaks in the third-order but this poses no problem to the security claims.

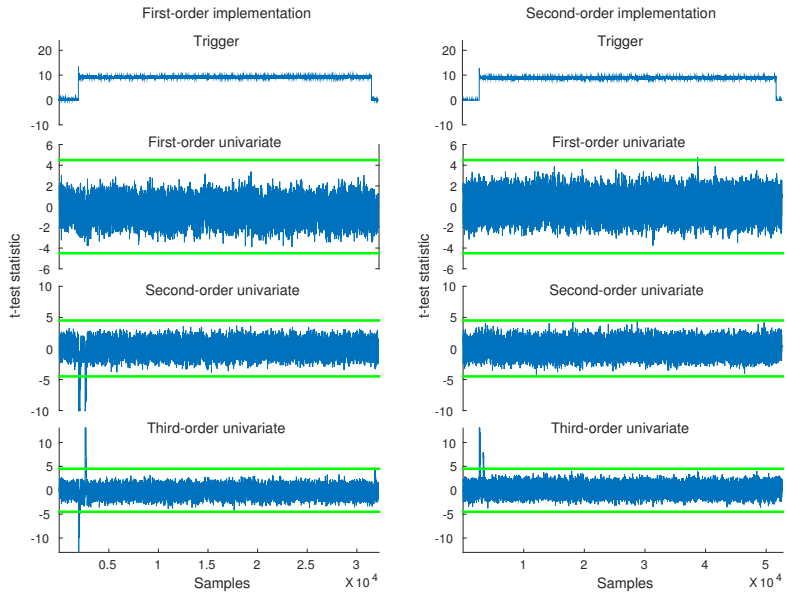

Fig. 3. Results of the first-, second- and third-order univariate analysis on the first- and second-order PRESENT S-box implementations.

We also performed a preliminary bivariate analysis. To this end, we preprocess each trace by first centering around a mean and then multiplying all possible pairs of time samples within a trace. This means that an m-sample trace is expanded into a $\binom{m}{2}$-sample trace, which results in a substantial increase in the computational and memory requirements. Then, a leakage test is performed on the preprocessed traces. To speed up the bivariate analysis, we opted for compressing the traces by a factor of 100.

As in the univariate case, we first carry out a sanity check to verify the soundness of this approach by performing a bivariate second-order analysis on the first-order secure implementation. This is expected to leak, and the results of Figure 4 confirm this. We obtained t-test statistic values within the region of

interest larger than 20, clearly indicating second-order bivariate leakage. These leakages are close to the diagonal, meaning that leakage occurs by combining samples from adjacent clock cycles. The leakage is visible with $200k$ traces.

We repeated the same experiment with the second-order secure implementation and found no value exceeding our confidence threshold of 4.5 with $1M$ traces. This provides some evidence that the second-order implementation indeed may be secure. However, we feel we cannot provide with a definite answer unless we exhaustively cover all possible pairs of time samples (without compression), something that is out of our current computational reach.

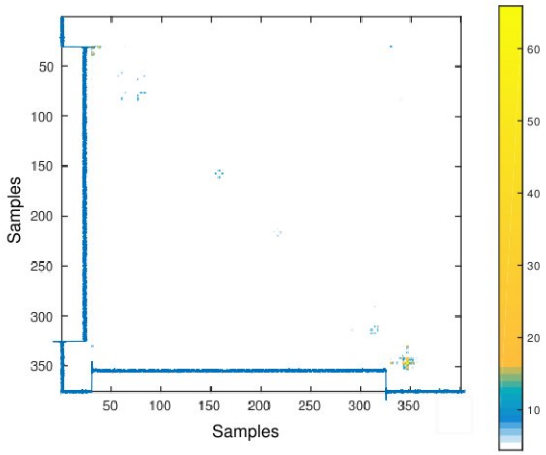

Fig. 4. Result of the second-order bivariate analysis on the first-order PRESENT S-box implementation.

5 Conclusions

We implemented a first- and second-order glitch resistant PRESENT S-box using the polynomial masking scheme presented in [22]. We verified these implementations with both univariate and bivariate attacks and confirmed the claimed SCA resistance. Our implementations resulted in 3594 GE for the first-order and in 8338 GE for the second-order implementation.

Acknowledgements

This work has been supported in part by the Research Council of KU Leuven (OT/13/071 and GOA/11/007), by the FWO (g.0550.12) and by the Hercules foundation (AKUL/11/19). Begül Bilgin was partially supported by the FWO project G0B4213N. Oscar Reparaz is funded by a PhD fellowship of the Fund for Scientific Research - Flanders (FWO).

References

1. Nangate open cell library. http://www.nangate.com/.
2. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM.
3. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
4. C. Carlet, L. Goubin, E. Prouff, M. Quisquater, and M. Rivain. Higher-Order Masking Schemes for S-Boxes. In A. Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012.
5. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
6. J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi. Test Vector Leakage Assessment (TVLA) methodology in practice. International Cryptographic Module Conference, 2013. http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf.
7. Y. Crama and P. L. Hammer. *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.
8. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. P.: A testing methodology for side-channel resistance validation, niat, 2011.
9. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side channel resistance validation. NIST non-invasive attack testing workshop, 2011. http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf.
10. L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In . K. Ko and C. Paar, editors, *CHES*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
11. J. Guo, T. Peyrin, and A. Poschmann. The PHOTON family of lightweight hash functions. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.
12. J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The LED block cipher. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.
13. ISO/IEC. *ISO/IEC 29192-2. Information technology – Security techniques – Lightweight cryptography – Part 2: Block ciphers*. ISO/IEC, 2012.
14. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.
15. S. Mangard, T. Popp, and B. M. Gammel. Side-channel leakage of masked CMOS gates. In A. Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
16. S. Mangard, N. Pramstaller, and E. Oswald. Successfully attacking masked AES hardware implementations. In J. R. Rao and B. Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

17. T. S. Messerges. Using second-order power analysis to attack dpa resistant software. In . K. Ko and C. Paar, editors, *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.
18. A. Moradi and O. Mischke. On the simplicity of converting leakages from multivariate to univariate - (case study of a glitch-resistant masking scheme). In *CHES*, pages 1–20, 2013.
19. S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In P. Ning, S. Qing, and N. Li, editors, *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.
20. A. Poschmann, S. Ling, and H. Wang. 256 bit standardized crypto for 650 GE - GOST revisited. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2010.
21. Research Center for Information Security, National Institute of Advanced Industrial Science and Technology. *Side-channel Attack Standard Evaluation Board SASEBO-G Specification*.
22. T. Roche and E. Prouff. Higher-order glitch free implementation of the AES using Secure Multi-Party Computation protocols - Extended version. *J. Cryptographic Engineering*, 2(2):111–127, 2012.
23. A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.
24. K. Tiri and I. Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. pages 246–251, 2004.
25. M. Tunstall and O. Benoît. Efficient use of random delays in embedded software. In D. Sauveron, C. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *WISTP*, volume 4462 of *Lecture Notes in Computer Science*, pages 27–38. Springer, 2007.
26. B. L. Welch. The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1/2):28–35, 1947.

Appendix A: Finite-field Multiplier

The combinatorial finite-field multiplier in $\mathbb{F}_{2^4}$ used in our implementation is based on the algebraic normal form. The 4-bit inputs $A = (a_3, a_2, a_1, a_0)$ and $B = (b_3, b_2, b_1, b_0)$ result in output $C = (c_3, c_2, c_1, c_0)$ by following bitwise operations:

$$
\begin{aligned}
c_0 &= (a_0b_0) + (a_1b_3) + (a_2b_2) + (a_3b_1) \\
c_1 &= (a_0b_1) + (a_1b_0) + (a_1b_3) + (a_2b_2) + (a_2b_3) + (a_3b_1) + (a_3b_2) \\
c_2 &= (a_0b_2) + (a_1b_1) + (a_2b_0) + (a_2b_3) + (a_3b_2) + (a_3b_3) \\
c_3 &= (a_0b_3) + (a_1b_2) + (a_2b_1) + (a_3b_0) + (a_3b_3)
\end{aligned}
$$

where A, B and C are in little-endian notation.

Appendix B: Polynomial Masking Scheme with (5,2)-sharing

This section lists the equations for the construction of, reconstruction from and operations on the shares when considering a (5,2)-sharing. In what follows, all additions and multiplications are in $\mathbb{F}_{2^4}$.

First, five distinct non-zero elements in $\mathbb{F}_{2^4}$ need to be chosen. These are referred to as the public coefficients $\alpha_{1\leq i\leq 5}$. Together with these points, the first row $(\lambda_1, ..., \lambda_5)$ of the inverse Vandermonde matrix $(\alpha_i^j)_{1\leq i,j\leq 5}$ is needed. These interpolation coefficients can be calculated as:

$$
\begin{aligned}
\lambda_1 &= \alpha_2(\alpha_1+\alpha_2)^{-1}\alpha_3(\alpha_1+\alpha_3)^{-1}\alpha_4(\alpha_1+\alpha_4)^{-1}\alpha_5(\alpha_1+\alpha_5)^{-1} \\
\lambda_2 &= \alpha_1(\alpha_2+\alpha_1)^{-1}\alpha_3(\alpha_2+\alpha_3)^{-1}\alpha_4(\alpha_2+\alpha_4)^{-1}\alpha_5(\alpha_2+\alpha_5)^{-1} \\
\lambda_3 &= \alpha_1(\alpha_3+\alpha_1)^{-1}\alpha_2(\alpha_3+\alpha_2)^{-1}\alpha_4(\alpha_3+\alpha_4)^{-1}\alpha_5(\alpha_3+\alpha_5)^{-1} \\
\lambda_4 &= \alpha_1(\alpha_4+\alpha_1)^{-1}\alpha_2(\alpha_4+\alpha_2)^{-1}\alpha_3(\alpha_4+\alpha_3)^{-1}\alpha_5(\alpha_4+\alpha_5)^{-1} \\
\lambda_5 &= \alpha_1(\alpha_5+\alpha_1)^{-1}\alpha_2(\alpha_5+\alpha_2)^{-1}\alpha_3(\alpha_5+\alpha_3)^{-1}\alpha_4(\alpha_5+\alpha_4)^{-1}
\end{aligned}
$$

Here, the multiplicative inverse in our field is represented by $.^{-1}$. Elements $\alpha_{1\leq i\leq 5}$ and $\lambda_{1\leq i\leq 5}$ are publicly available to all five players.

Sharing a value X requires two secret and random coefficients a_1, a_2 and the public coefficients $\alpha_{1\leq i\leq 5}$. The resulting shares $X_{1\leq i\leq 5}$ are calculated as:

$$X_i = X + (a_1\alpha_i) + (a_2\alpha_i^2), \text{ with } 1 \leq i \leq 5$$

Each player receives exactly one share X_i and has no access to any other share.

Reconstruction of the secret value X requires the interpolation coefficients $\lambda_{1\leq i\leq 5}$:

$$X = (X_1\lambda_1) + (X_2\lambda_2) + (X_3\lambda_3) + (X_4\lambda_4) + (X_5\lambda_5)$$

To describe the operations, a constant value will be represented as c and two secret values as X and Y. Their (5,2)-sharings are given by $X_{1\leq i\leq 5}$ and $Y_{1\leq i\leq 5}$. Both are masked with the same public coefficients but use independent random secret coefficients a_1, a_2 and b_1, b_2.

Addition with a constant can be achieved by each player independently as:

$$\begin{aligned} Z_i = &\ X_i + c \\ &= (X + (a_1\alpha_i) + (a_2\alpha_i^2)) + c \\ &= (X + c) + (a_1\alpha_i) + (a_2\alpha_i^2), \text{ with } 1 \leq i \leq 5 \end{aligned}$$

The resulting shares of the addition represent the correct new secret $Z = X + c$.

Multiplication with a constant is performed in a similar way and can again be achieved by each player independently:

$$\begin{aligned} Z_i = &\ X_i c \\ &= (X + (a_1\alpha_i) + (a_2\alpha_i^2))c \\ &= (Xc) + (a_1 c\alpha_i) + (a_2 c\alpha_i^2), \text{ with } 1 \leq i \leq 5 \end{aligned}$$

Considering $(a_1\ c)$ and $(a_2\ c)$ as the new coefficients of the second-order polynomial, the shares $Z_{1\leq i\leq 5}$ represent the desired output $Z = Xc$. Note that the reconstruction of the masked secret variable does not depend on the polynomial coefficients a_1, a_2, but on the interpolation coefficients $\lambda_{1\leq i\leq 5}$, which only depend on the public coefficients $\alpha_{1\leq i\leq 5}$.

Addition of two shared secrets is executed in following way:

$$\begin{aligned} Z_i = &\ X_i + Y_i \\ &= (X + (a_1\alpha_i) + (a_2\alpha_i^2)) + (Y + (b_1\alpha_i) + (b_2\alpha_i^2)) \\ &= (X + Y) + (a_1 + b_1)\alpha_i) + (a_2 + b_2)\alpha_i^2), \text{ with } 1 \leq i \leq 5 \end{aligned}$$

With $a_1 + b_1$ and $a_2 + b_2$ as the new polynomial coefficients, the resulting shares mask the desired new secret variable $Z = X + Y$.

Multiplication of two shared secrets consists of the following three steps:

1. Each player i first computes t_i

$$\begin{aligned} t_i &= X_i Y_i \\ &= (XY) + (a_1 Y + b_1 X)\alpha_i + (a_1 b_1 + a_2 Y + b_2 X)\alpha_i^2 \\ &\quad + (a_1 b_2 + b_1 a_2)\alpha_i^3 + (a_2 b_2)\alpha_i^4, \text{ with } 1 \leq i \leq 5 \end{aligned}$$

2. Each player i then randomly selects two coefficients $a_{i,1}$, $a_{i,2}$ and remasks t_i:

$$\begin{aligned} q_{i,1} &= t_i + (a_{i,1}\alpha_1) + (a_{i,2}\alpha_1^2) \\ q_{i,2} &= t_i + (a_{i,1}\alpha_2) + (a_{i,2}\alpha_2^2) \\ q_{i,3} &= t_i + (a_{i,1}\alpha_3) + (a_{i,2}\alpha_3^2) \\ q_{i,4} &= t_i + (a_{i,1}\alpha_4) + (a_{i,2}\alpha_4^2) \\ q_{i,5} &= t_i + (a_{i,1}\alpha_5) + (a_{i,2}\alpha_5^2) \end{aligned}$$

   Each $q_{i,\forall j \neq i}$ is subsequently send to the corresponding player j.
3. The outputs $q_{1,i}$, $q_{2,i}$, $q_{3,i}$ of each player i are then distributed and reconstructed as

$$Z_i = (q_{1,i}\lambda_1) + (q_{2,i}\lambda_2) + (q_{3,i}\lambda_3) + (q_{4,i}\lambda_4) + (q_{5,i}\lambda_5)$$

This sequence of operations gives the shares corresponding to the correct masked result $Z = XY$ in a secure way.

Square of a shared secret can only be computed in the straightforward way, i.e., as $Z = X^2$ or

$$Z_i = Z_i^2 = X^2 + (a_1^2\alpha_i^2) + (a_2^2(\alpha_i^2)^2), \text{ with } 1 \leq i \leq 5$$

when $\alpha_{1 \leq i \leq 5}$ satisfy the conditions for frobenious stability. This means that for every α_i, there exists an α_j such that $\alpha_j = \alpha_i^2$. A reordering between every player i and player j satisfying $\alpha_j = \alpha_i^2$ is then required to keep the correct public coefficient linked to its player. When this reordering is not performed, the reconstruction of the correct masked secret $Z = X^2$ is not possible.

Appendix C: Second-order Hardware Architecture

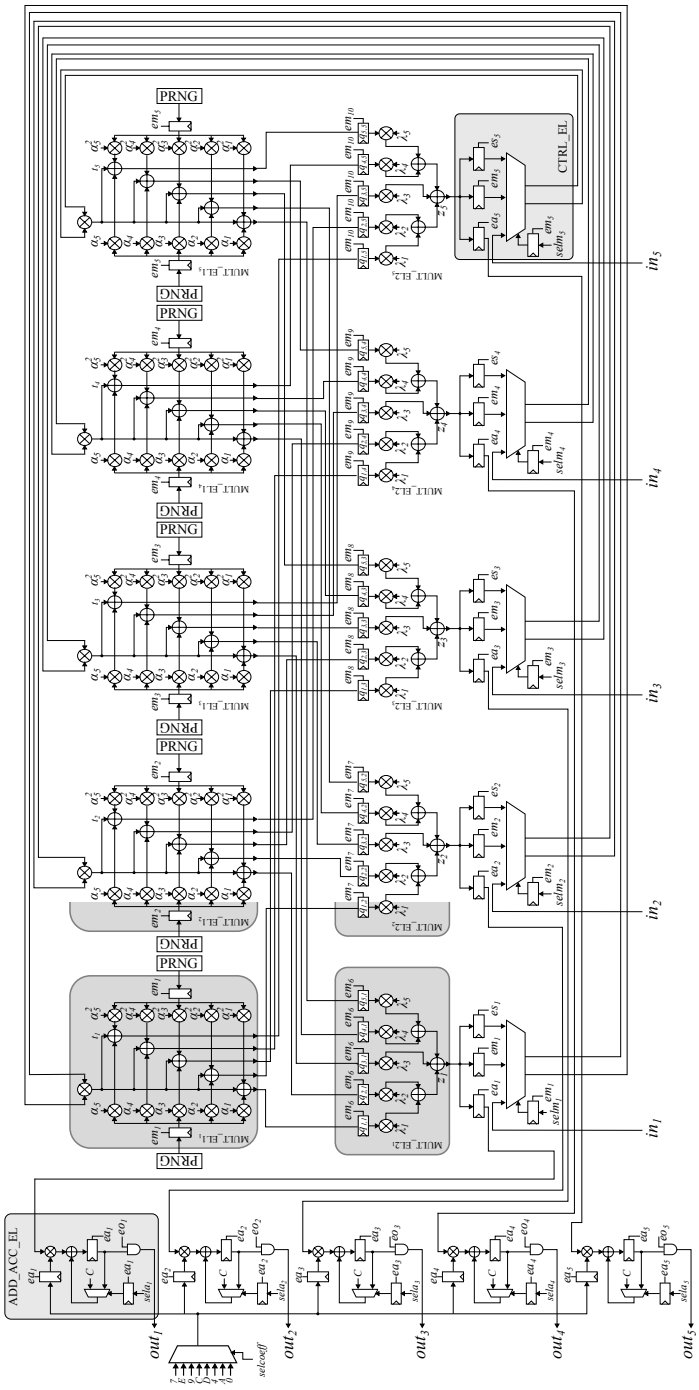

Fig. 5. Architecture diagram for the second-order PRESENT implementation.

Appendix D: Area requirements for the first-order and second-order present S-box implementations

Table 3. Area in GE of the first-order and second-order PRESENT S-box implementations.

Component	Area (GE)	
	first-order	second-order
multiplier	47	47
mult_el1	233	639
mult_el2	148	252
shared_mult	1360	4969
add_acc_el	127	127
add_acc	379	630
ctrl_el	120	120
ctrl	352	592
Control unit	1503	2147
S-box	3594	8338

Appendix E: Welch's t-Test

An easy way to test for potential side-channel leakages, which might lead to a successful attack in a cryptographic system, is proposed by Goodwill *et al.* [8]. Due to its independence of a leakage model. this method is a convenient way to test whether or not the implementation of the device effectively counteracts SCA attacks. Although no single test can guarantee the revelation of all vulnerabilities against all possible SCA attacks, this test is designed to be sensitive enough to cover a wide range of potential problems. After acquisition of a sufficient amount of power traces, the traces are divided in two sets, A and B, based on an intermediate value in the computation. The problem of assessing whether there is potentially exploitable leakage or not is then formulated as an hypothesis test. The null hypothesis corresponds to the statement "the mean power curves of A and B are data-independent". The statistical test is *Welch's t-test*, a generalization of the *Student's t-test* allowing samples to have unequal variances [26]. For the first statistical moment, the t-test statistic is calculated as:

$$t = \frac{\overline{T_a} - \overline{T_b}}{\sqrt{\frac{s_a^2}{N_a} + \frac{s_b^2}{N_b}}}$$

where $\overline{T_i}$, s_i^2, N_i are the sample mean, sample variance and sample size of the set $T_{i \in a,b}$. This formula can easily be extended to higher statistical moments.

The t-test statistic is computed point-wise on the different sets of power traces. If no point exceeds a certain confidence threshold $\pm C$, then the null hy-

pothesis holds, indicating that there is no relation between the processed intermediate value and the instantaneous power consumption. In case the threshold is crossed, another t-test is performed on an independent set of traces. When the t-test statistic exceeds $\pm C$ at the same points in time, the null hypothesis can be rejected with a significance level related to C. In that case, the alternate hypothesis holds, indicating that the power consumption and the intermediate values are related in a statistically significant way, making the device potentially vulnerable to SCA attacks.

Figure 6 shows the resulting t-test statistic in case the alternate hypothesis hold.

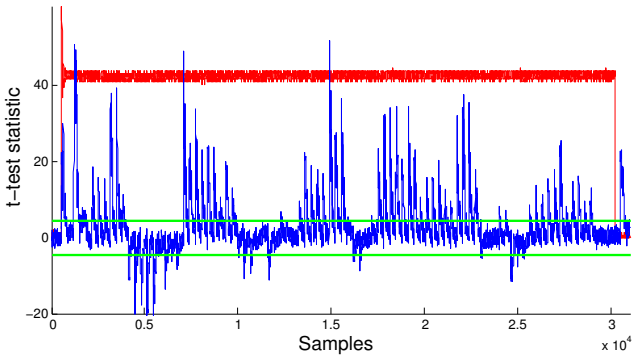

Fig. 6. Result of the t-test for the first-order PRESENT implementation with biased masks.